N91-20684

A Failure Management Prototype: DR/Rx

David G. Hammen
Carolyn G. Baker
Christine M. Kelly
Christopher A. Marsh

The MITRE Corporation
1120 NASA Road One
Houston, Texas 77058

Abstract

This failure management prototype performs failure diagnosis and recovery management of hierarchical, distributed systems. The prototype, which evolved from a series of previous prototypes following a spiral model for development, focuses on two functions: the Diagnostic Reasoner (DR) performs integrated failure diagnosis in distributed systems, and the Recovery Expert (Rx) develops plans to recover from the failure. This paper discusses issues related to expert system prototype design, discusses the previous history of this prototype, and describes the architecture of the current prototype in terms of the knowledge representation and functionality of its components.

Introduction

Space Station Freedom has been defined to have a hierarchical, distributed control architecture. The highest level in the architecture, Tier I, has knowledge of each of the systems in Freedom (for example, the Communications and Tracking System (C&TS) and the Thermal Control System (TCS)). The Operations Management System (OMS), composed of both automated functions and manual operations, represents Tier I in the command architecture. The second level, Tier II, represents a lower level in the command hierarchy, having a limited scope of knowledge (for example, the System Management function for the Electrical Power System, which has little or no knowledge of the other systems). Tier II managers' functions are further delegated to Tier III managers. The data become more abstract and qualitative as they advance upward through the control hierarchy.

This paper describes a prototype* that is being designed to perform failure management at the Tier I (OMS) level. Failure management includes diagnosing the failure, determining the corrective actions to take, and then taking the actions and tracking the progress of the recovery. The first phase of the project implements the first two of these three functions: the Diagnostic Reasoner (DR) performs diagnosis and the Recovery Expert (Rx) establishes a Course of Action to take to effect recovery.

This paper discusses issues related to expert system prototype design, discusses the previous history of the current prototype, and describes the architecture of this prototype in terms of the knowledge representation and functionality of its components.

Related Works

Our current effort expands on previous work done by others and by ourselves. Our current prototype expands on our previous efforts (Marsh, 1988; Marsh, 1989) by greatly increasing the use of behavior representation, by addressing the impacts that result from a failure, and by developing plans to recover from a failure. The Diagnostic Reasoner incorporates research from model-based reasoning, focusing on the works of Davis (Davis, 1985), de Kleer and Williams (de Kleer, 1987), Geffner and Pearl (Geffner, 1987), and Holtzblatt, Marcotte and Piazza (Holtzblatt, 1989). The Recovery Expert incorporates research from planning, focusing on the goal-directed planning developments by Wilkins (Wilkins, 1988) and the

* The research on and development of this prototype was jointly sponsored by the National Aeronautics and Space Administration, Johnson Space Center, under contract NAS9-18057, and by The MITRE Corporation under its MITRE Sponsored Research Program.

Procedural Reasoning System described by Georgeff and Ingrand (Georgeff, 1989).

Design Methodology

Design techniques used to build knowledge-based expert systems are quite different from those used to develop conventional software systems. Conventional software systems are developed using principles of modern software engineering, while expert systems development follows knowledge engineering disciplines.

Software systems developed using the waterfall model follow well-defined design methodologies and techniques and procedures that support them. This allows the project manager to control the software development process; the developer is provided a foundation for building high-quality software in a productive manner (Pressman 1987). NASA has baselined the use of the waterfall model for the development of software for the Space Station Freedom Program (NASA 1989).

A pitfall to avoid when following this method is its over emphasis on fully-elaborated documents in the early design phase at the expense of attention to functionality and meeting the user needs. The waterfall model is appropriate where budget and schedule are the primary concerns, but is ill-suited when good user interfaces and decision support aids functions are required (Boehm, 1988).

In developing a knowledge-based expert system, the phases of the development process are interleaved. The capabilities of the product evolve as a function of operating experience. This technique is well suited to knowledge-based applications where the concepts are not well known at the start of the project. This promises a rapid initial operating capability from which the product can evolve (Boehm, 1988). The key tasks for the developer are to gather domain knowledge from an expert, build a portion of the system, and then work with the expert to refine the product (Waterman, 1986).

A pitfall to avoid when using the iterative methodology is a tendency to incorporate additional capabilities that exceed the initial design assumptions and constraints; the resulting product is no longer an integrated piece of software but a large and unruly collection of routines and constructs. At this point, the design should be re-assessed and the system re-implemented to improve the conceptualization of the existing knowledge, if the system is to continue to grow in depth and breadth (Hayes-Roth, 1983).

The spiral model proposed by Boehm (Boehm, 1988) describes a development spiral in which concepts are discovered, implemented as prototypes and evaluated. The prototypes are discarded, but the valid concepts are retained and re-implemented in a more refined product. The spiral model provides for product life-cycle evolution and growth and focuses on identifying and resolving risk items.

History

The past history for the evolution of the OMS prototype has largely followed the spiral model, with ideas being re-implemented as operating concepts have matured. This paper describes the current phase in the prototype life cycle in which new ideas are being added, and some previous work is being re-implemented to reflect a closer-to-operations environment.

The first prototypes, implemented in a Lisp environment, demonstrated the use of inferencing in failure diagnosis and the use of automation in activity execution and monitoring. Eventually, the first prototypes were integrated on a test bed with simulation of Space Station Freedom systems (Marsh, 1988).

Once the test bed environment matured, it was necessary to refine on the capabilities of the first prototypes and re-implement them based on test bed operational constraints. A combination of C and Ada were used for this phase of implementation (Marsh, 1989; Kelly, 1989).

This paper describes the next step in the prototype evolution. An additional capability (Rx, to plan for recovery) is being added and integrated with DR, the failure diagnosis component. DR, a re-implementation of failure diagnosis, contains earlier diagnostic capabilities, but expands on the use of models, both to support diagnosis and to assist in the planning for recovery. Eventually, these two components will be integrated with the execution of the activities identified to effect recovery.

OMS Prototype Design

The OMS failure management prototype will be implemented in Ada (the language mandated for the Space Station Freedom Program) and ART/Ada (an

expert system shell) on a VAX Station 3100 workstation under the VMS operating system.

We will continue using the spiral development methodology for this effort. Some iteration is required for the development of our knowledge and model bases as Freedom's design is subject to change. Iteration is also required for the development of our application software as the exact techniques to use or avoid are not yet well-known. The spiral methodology embodies this need for iteration.

The spiral method can support the complexity of the OMS design and provides the rigor necessitated by early definition of Ada specifications and interfaces and Ada's emphasis on strong typing. Compared to the very large projects developed using the waterfall model, the OMS prototype and development team are quite small; the extensive project management, configuration management, and documentation required by software engineering are not necessary for our small prototyping effort.

The design of two of the prototype's functions, the DR and Rx, has been recently completed. DR determines likely failure sources and their potential impacts and Rx develops plans to recover from these failures. An overview of the information flow through DR and Rx processes is presented in figure 1. This figure introduces a hypothetical scenario that relies in part on interactions between a C&TS frame multiplexer and a TCS cold plate; this scenario will be used to illustrate the design. Discussions of the design of these two functions follow.

DR Architecture

The DR is responsible for determining the likely source(s) of a failure and synthesizing dynamic summary failure reports from the Tier II systems. The DR's diagnosis could confirm or correct system-level diagnoses.

Updating the Component Model

The Tier II managers notify the DR of changes in the status of system components and changes in the relationships between those components through System Reports. Only significant qualitative changes

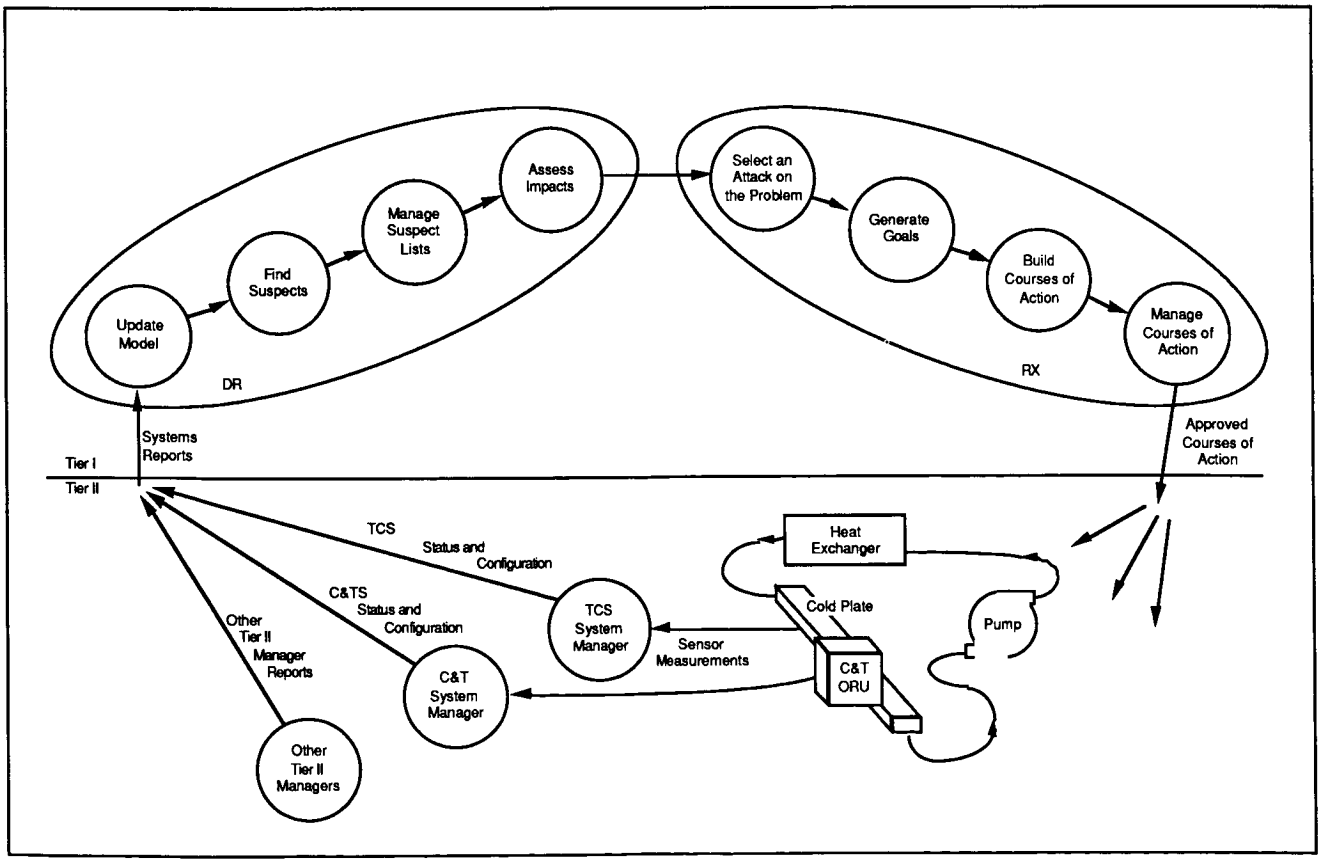

Figure 1 Overview of DR/Rx Process Flow

(for example, a cold plate's temperature changing from "nominal" to "hot") are reported by the systems to DR; minor quantitative deviations (for example, the cold plate's temperature changing from 70 to 72 degrees) are not. DR uses the information in the System Reports to update a schematic-like model of the Space Station Freedom's systems.

The component model incorporates configuration, status and behavior information. The configuration information identifies a component's relationships with other components, both physically and functionally. The status information identifies mode of operation, equipment health, and key operational measures that are related to behavior. The behavior information identifies the causes and effects of particular conditions with respect to a particular component's health and mode of operation. The behavior information describes both internal causal consequences and behaviors across configuration boundaries.

The description of a specific component is based on the generic description of a class of related components; the class descriptions in turn could be defined as a hierarchy of descriptions. The description of a class of components includes descriptions of behavior causes and effects and attribute definitions of behavior measure and configuration elements. The description of a specific component includes information about the component's behavior measures, operating conditions, and configuration. A portion of the Component Model is depicted in figure 2.

Generating a Suspect List

When the System Reports indicate a problem (as opposed to a nominal change in configuration), DR determines suspected causes based on reported behavior and modeled behavioral cause-and-effect. This information is collected into a Suspect List. DR verifies that the expected behavioral effects have occurred with respect to the possible suspects and their related components.

A Suspect List identifies those suspects that will result in a particular set of observed behaviors. More than one cause could result in same observable behaviors, in which case DR will identify each possible cause as a possible suspect in the same Suspect List. A set of observable behaviors could result from multiple component failures, in which case DR will identify the failure group as a possible cause. A Suspect List also could identify key unknown behavior measures: the assessment of some behavior measures will require special resources or will induce inter-system interactions. When DR encounters one of these unknown measures along one of its diagnostic causal pathways it will post the measure in the Suspect List as a key unknown behavior measure whose assessment should help refine the diagnosis.

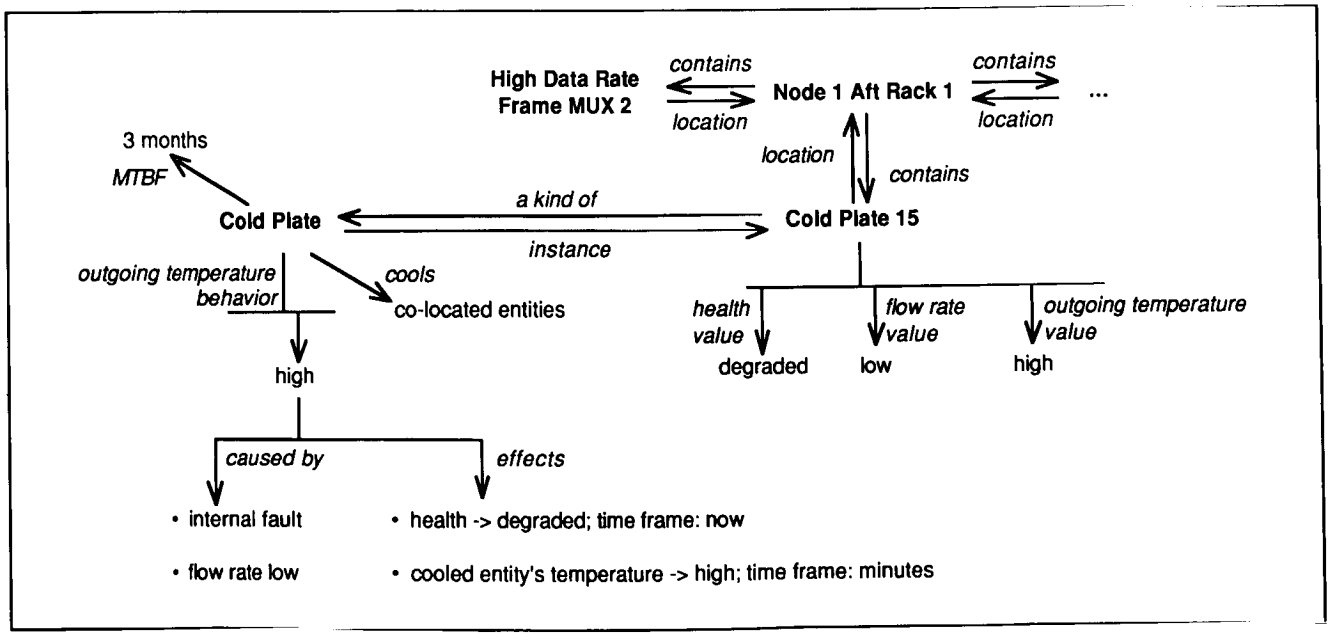

Figure 2 Component Model

Managing Suspect Lists

The Tier II managers do not observe all of the effects of a failure at one time. Consequently, DR will generate Suspect Lists without complete knowledge of the problem. When DR receives the first System Report, DR responds given the available information. As additional information becomes available, DR relies on the expected behavioral effects or explained behaviors of identified suspects that are described in the component model to merge Suspect Lists generated as the result of the same failure.

Assessing Impacts

The suspected components that produce the most immediate and most critical impacts should be considered before those suspects which have less severe consequences. To assess the impact of a particular suspect, DR uses an Impact Model that augments the Component Model. The Impact Model focuses on cascading operational causes and effects and looks further ahead in time than does the Component Model. To help assess the significance of an impact, the Impact Model heuristically assigns numerical severity values to an impact. A severity value and temporal factor are attached to each node in the derived, suspect-specific Impact Sequence. A sample impact sequence is presented in figure 3.

Rx Architecture

The Rx is responsible for determining and recommending a set of procedures, based on the available crew procedures, that will result in recovery from the problem. This set of procedures could include intermediate actions that mitigate the more acute consequences of the problem, providing adequate time to realize the recovery itself.

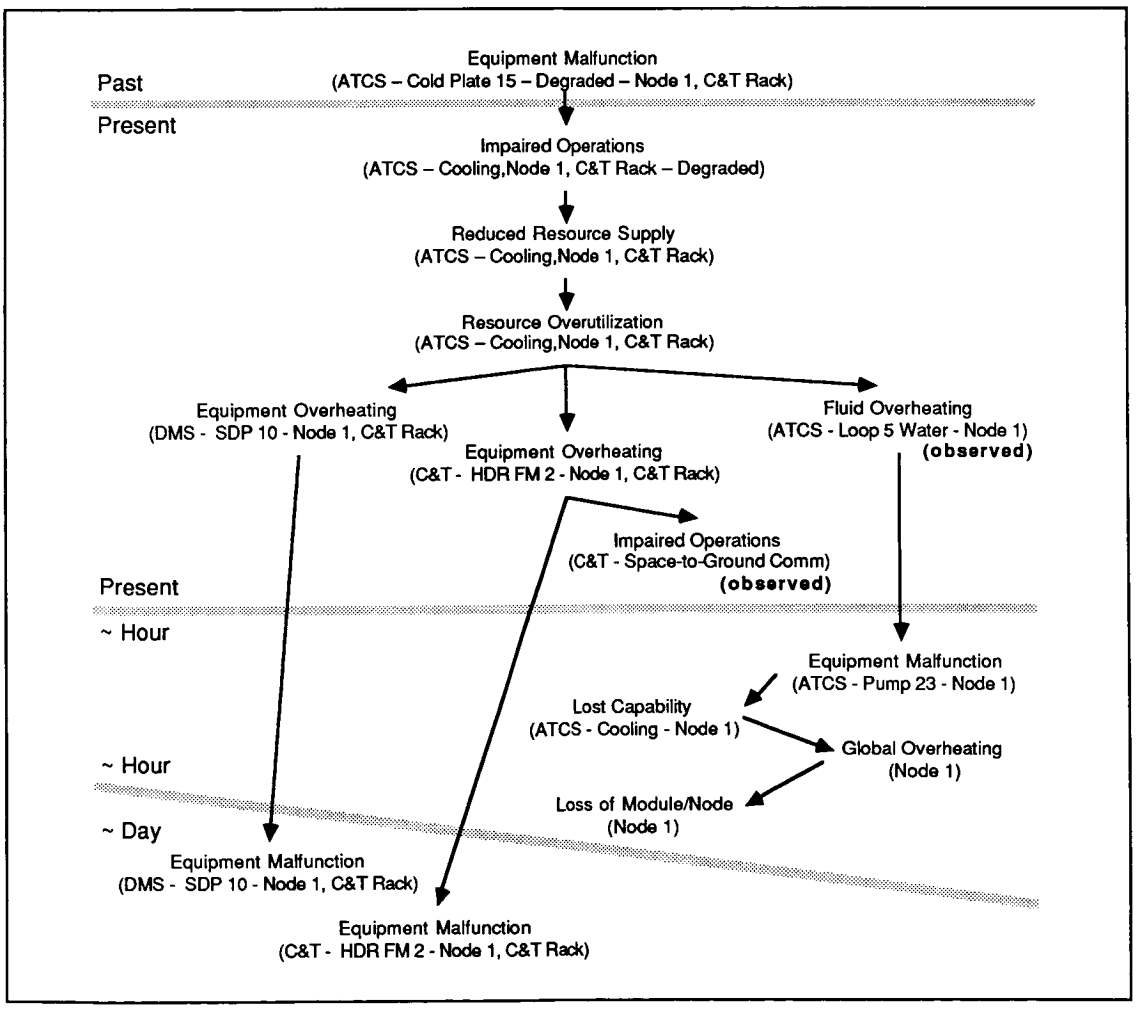

Figure 3 Impact Sequence

Selecting an Attack

Several options are available for dispositioning the diagnosis reported by DR. Some failures are accurately identified by DR; in these cases, the problem can be addressed directly. Other failures are not easily identified; additional information is needed, such as information provided by an inter-system diagnostic test procedure or a behavior measure value that is unknown but whose assessment involves inter-system interactions (and therefore the value cannot be determined without Tier I approval). Rx is also concerned with assuring that the actions taken are sensible in light of the foreseeable impacts. When severe or acute downstream impacts could occur, Rx will develop plans that mitigate these downstream impacts so that the desired action can be sensibly performed. It might be imprudent for Rx to initiate any action when a preliminary Suspect List is reported by DR: Rx might do nothing until DR has observed some predicted near-term downstream impacts. Finally, Rx will request operator intervention if it cannot find an adequate response.

Generating Goals

Rx develops goals that address the failure in concert with the chosen attack. For example, repair goals directly address failures, impact mitigation goals address downstream impacts, data collection goals address unknown behavior measures, and diagnostic goals address unclear diagnoses. These goals will drive the generation of a plan to solve a specific part of the overall problem.

Rx applies generic goals that address the selected attack to the specific problem, forming a specific goal that addresses the specific problem and the selected attack. For example, Rx could address a reduction in cooling capacity as a special case of a resource supply reduction. The generic goal of reducing resource consumption addresses this generalized problem. Applying this generic goal is applied to the specific problem results in the specific goal of reducing the cooling load. Sample goal generation data for impact mitigation are presented in table 1.

Building Courses of Action

A Course of Action specifies a set of procedures that collectively achieve a specific goal. The procedures are selected from the pre-defined set of flight procedures. Procedure metadata describes reasons for, outcomes of and constraints against the use of procedures. Rx uses this procedure metadata to build a Course of Action that achieves the specific goals within the constraints imposed by the failure and its impacts. The Course of Action specifies the names of procedures and the temporal relationships between them: the procedures, when executed, should achieve the specific goal that the Course of Action addresses.

Managing Courses of Action

Rx can build multiple Courses of Action in response to a single problem. For example, the desired action should be achieved by a repair Course of Action, but several impact mitigation Courses of Action will be required for this desired action to have a successful outcome. These multiple Courses of Action must be merged and ordered to form a unified Course of Action that addresses the entire problem rather than a portion of the problem. The attempt is to build Courses of Action that solve the root problem within the constraints levied by the failure. Sample Courses of Action are presented in table 2.

Rx can also develop alternate means of addressing the problem. These alternate Courses of Action must be evaluated prior to execution. These final steps, as well as all other steps in the process, require crew interaction and approval.

Conclusions

The design of the DR and Rx has been recently completed. This design was achieved by using software engineering and knowledge engineering techniques. These techniques can be merged under the spiral model for a more integrated and long-lived system. This design incorporates some concepts from the predecessor prototyping activities but also introduces some new ideas.

Future Plans

A Procedures Interpreter was a previous product of the evolution of this prototype. This will be folded into our current work to execute the recommended Course of Action and to ensure that this Course of Action is achieving the desired goals. The current prototype is a stand-alone product and will be integrated into the test bed environment to demonstrate the effectiveness and an integrated failure management system.

Table 1
Goal Generation Information for Impact Mitigation

Impact	Workarounds	Goal Generation Information
Impaired Operations	Use Backup Capability	Backup Capability Mode On and Failed Equipment Mode Off
Reduced Resource Supply	Augment Resource Supply	Resource Level Nominal
Resource Overutilization	Augment Resource Supply or Decrease Resource Utilization	Resource Level Nominal Resource Consumption <= Resource Level

Table 2
Courses of Action

Goal	Course of Action	Comments
Repair Cold Plate	Repair Cold Plate 15	Severe impacts occur before completion
Reduce Cold Plate Load	Cross-Strap Frame Multiplexer 2	Does not attack root problem
Reduce Cold Plate Load and Repair Cold Plate	Cross-Strap Frame Multiplexer 2 and Repair Cold Plate 15	Timely and effective

References

Boehm, Barry W. (May 1988) A Spiral Model of Software Development and Enhancement. *IEEE Computer*, Volume 21, Number 5, pp 61-72.

Davis, Randall (1985), *Diagnostic Reasoning Based on Structure and Behavior*, Qualitative Reasoning About Physical Systems, Daniel G. Bobrow, ed, Cambridge, Massachusetts: The MIT Press, pp 347-410.

De Kleer, Johan and Brian C. Williams (1987), *Diagnosing Multiple Faults*, Artificial Intelligence, Volume 32, Number 1, pp 97-130.

Geffner, Hector and Judea Pearl (1987), An Improved Constraint-Propagation Algorithm for Diagnosis, Proceedings of the Tenth International Joint Conference on Artificial Intelligence, San Mateo, California: Morgan Kaufmann Publishers, Inc., pp 1105-1111.

Georgeff, Michael P. and François Felix Ingrand (1989), *Decision-Making in Embedded Reasoning Systems*, Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, San Mateo, California: Morgan Kaufmann Publishers, Inc., pp 972-978.

Holtzblatt, L. J., R. A. Marcotte and R.L. Piazza (October 1989), *Overcoming Limitations of Model-based Diagnostic Reasoning Systems*, presented at the AIAA Computers in Aerospace VII Conference.

Hayes-Roth, Frederick, Donald A. Waterman and Douglas B. Lenat, eds. (1983), *Building Expert Systems*, Reading, Massachusetts: Addison-Wesley Publishing Company Inc.

Kelly, Christine, Christopher Marsh, Alain Jouchoux and Fred Lacy (1989), *The Migration of an Expert System from Lisp to Ada*, Proceedings of AIDA-89: Fifth Annual Conference on Artificial Intelligence and Ada, Fairfax, Virginia: George Mason University, pp 108-119.

Marsh, Christopher (1988), *The ISA Expert System: A Prototype System for Failure Diagnosis on the Space Station*, Proceedings of the First International Conference on Industrial Engineering Applications of Artificial Intelligence and Expert Systems, New York, New York: ACM Press, pp 60-74.

Marsh, Christopher and Christine Kelly (1989), *Operations Management Application Prototype*, Fourth Artificial Intelligence and Simulation Workshop, Detroit, Michigan, pp 75-77.

Pressman, Roger S. (1987), Software Engineering: A Practitioner's Approach, New York, New York: McGraw-Hill.

NASA (February 1989), *Information System Life Cycle and Documentation Standards, Rel 4.3*, Washington D.C.: NASA Office of Safety, Reliability, Maintainability, and Quality Assurance–Software Management and Assurance Program.

Waterman, Donald A. (1986), *A Guide to Expert Systems*, Reading, Massachusetts: Addison-Wesley Publishing Company Inc.

Wilkins, David E. (1988), *Practical Planning: Extending the Classical AI Planning Paradigm*, San Mateo, California: Morgan Kaufmann Publishers, Inc.